Kotlin - Lists

Kotlin list is an ordered collection of items. A Kotlin list can be either mutable (**mutableListOf**) or read-only (**listOf**). The elements of list can be accessed using indices. Kotlin mutable or immutable lists can have duplicate elements.

Creating Kotlin Lists

For list creation, use the standard library functions **listOf()** for read-only lists and **mutableListOf()** for mutable lists.

> To prevent unwanted modifications, obtain read-only views of mutable lists by casting them to List.

Example

```
fun main() {
    val theList = listOf("one", "two", "three", "four")
    println(theList)

    val theMutableList = mutableListOf("one", "two", "three", "four")
    println(theMutableList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[one, two, three, four]
[one, two, three, four]
```

AD

Loop through Kotlin Lists

There are various ways to loop through a Kotlin list. Lets study them one by one:

Using toString() function

```
fun main() {
    val theList = listOf("one", "two", "three", "four")
    println(theList.toString())
}
```

When you run the above Kotlin program, it will generate the following output:

```
[one, two, three, four]
```

Using Iterator

```kotlin
fun main() {
    val theList = listOf("one", "two", "three", "four")

    val itr = theList.listIterator()
    while (itr.hasNext()) {
        println(itr.next())
    }
}
```

When you run the above Kotlin program, it will generate the following output:

```
one
two
three
four
```

Using for loop

```kotlin
fun main() {
   val theList = listOf("one", "two", "three", "four")

   for (i in theList.indices) {
      println(theList[i])
   }
}
```

When you run the above Kotlin program, it will generate the following output:

```
one
two
three
four
```

Using forEach

```kotlin
fun main() {
   val theList = listOf("one", "two", "three", "four")

   theList.forEach { println(it) }
}
```

When you run the above Kotlin program, it will generate the following output:

```
one
two
```

```
three
four
```

> Note - here **it** works like **this** operator in Java.

Size of Kotlin List

We can use **size** property to get the total number of elements in a list:

```
fun main() {
    val theList = listOf("one", "two", null, "four", "five")

    println("Size of the list " + theList.size)
}
```

When you run the above Kotlin program, it will generate the following output:

```
Size of the list 5
```

The "in" Operator

The **in** operator can be used to check the existence of an element in a list.

Example

```
fun main() {
   val theList = listOf("one", "two", "three", "four")

   if("two" in theList){
      println(true)
   }else{
      println(false)
   }
}
```

When you run the above Kotlin program, it will generate the following output:

```
true
```

The contain() Method

The **contain()** method can also be used to check the existence of an element in a list.

Example

```
fun main() {
   val theList = listOf("one", "two", "three", "four")

   if(theList.contains("two")){
      println(true)
   }else{
      println(false)
   }

}
```

When you run the above Kotlin program, it will generate the following output:

```
true
```

The isEmpty() Method

The **isEmpty()** method returns **true** if the collection is empty (contains no elements), **false** otherwise.

Example

```
fun main() {
   val theList = listOf("one", "two", "three", "four")

   if(theList.isEmpty()){
      println(true)
   }else{
      println(false)
   }
}
```

When you run the above Kotlin program, it will generate the following output:

```
false
```

The indexOf() Method

The **indexOf()** method returns the index of the first occurrence of the specified element in the list, or -1 if the specified element is not contained in the list.

Example

```
fun main() {
   val theList = listOf("one", "two", "three", "four")

   println("Index of 'two' :  " + theList.indexOf("two"))
}
```

When you run the above Kotlin program, it will generate the following output:

```
Index of 'two' : 1
```

The get() Method

The **get()** method can be used to get the element at the specified index in the list. First element index will be zero.

Example

```
fun main() {
   val theList = listOf("one", "two", "three", "four")

   println("Element at 3rd position " + theList.get(2))
}
```

When you run the above Kotlin program, it will generate the following output:

```
Element at 3rd position three
```

List Addition

We can use **+** operator to add two or more lists into a single list. This will add second list into first list, even duplicate elements will also be added.

Example

```
fun main() {
    val firstList = listOf("one", "two", "three")
    val secondList = listOf("four", "five", "six")
    val resultList = firstList + secondList

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[one, two, three, four, five, six]
```

List Subtraction

We can use **-** operator to subtract a list from another list. This operation will remove the common elements from the first list and will return the result.

Example

```
fun main() {
    val firstList = listOf("one", "two", "three")
    val secondList = listOf("one", "five", "six")
    val resultList = firstList - secondList

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[two, three]
```

Slicing a List

We can obtain a sublist from a given list using **slice()** method which makes use of range of the elements indices.

Example

```
fun main() {
    val theList = listOf("one", "two", "three", "four", "five")
    val resultList = theList.slice( 2..4)

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[three, four, five]
```

Removing null a List

We can use **filterNotNull()** method to remove **null** elements from a Kotlin list.

```
fun main() {
    val theList = listOf("one", "two", null, "four", "five")
    val resultList = theList.filterNotNull()

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[one, two, four, five]
```

Filtering Elements

We can use **filter()** method to filter out the elements matching with the given predicate.

```
fun main() {
    val theList = listOf(10, 20, 30, 31, 40, 50, -1, 0)
    val resultList = theList.filter{ it > 30}

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[31, 40, 50]
```

Dropping First N Elements

We can use **drop()** method to drop first N elements from the list.

```
fun main() {
    val theList = listOf(10, 20, 30, 31, 40, 50, -1, 0)
    val resultList = theList.drop(3)

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[31, 40, 50, -1, 0]
```

Grouping List Elements

We can use **groupBy()** method to group the elements matching with the given predicate.

```
fun main() {
    val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    val resultList = theList.groupBy{ it % 3}

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
{1=[10, 31, 40], 0=[12, 30, 9, -3, 0]}
```

Mapping List Elements

We can use **map()** method to map all elements using the provided function:.

```
fun main() {
    val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    val resultList = theList.map{ it / 3 }

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[3, 4, 10, 10, 13, 3, -1, 0]
```

Chunking List Elements

We can use **chunked()** method to create chunks of the given size from a list. Last chunk may not have the elements equal to the number of chunk size based on the total number of elements in the list.

```
fun main() {
    val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    val resultList = theList.chunked(3)

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[[10, 12, 30], [31, 40, 9], [-3, 0]]
```

Windowing List Elements

We can use **windowed()** method to a list of element ranges by moving a sliding window of a given size over a collection of elements.

```
fun main() {
    val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    val resultList = theList.windowed(3)

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[[10, 12, 30], [12, 30, 31], [30, 31, 40], [31, 40, 9], [40, 9, -3], [9, -3, 0]]
```

By default, the sliding window moves one step further each time but we can change that by passing a custom step value:

```
fun main() {
    val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    val resultList = theList.windowed(3, 3)

    println(resultList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[[10, 12, 30], [31, 40, 9]]
```

Kotlin mutable List

We can create mutable list using **mutableListOf()**, later we can use **add()** to add more elements in the same list, and we can use **remove()** method to remove the elements from the list.

```
fun main() {
    val theList = mutableSetOf(10, 20, 30)

    theList.add(40)
    theList.add(50)
    println(theList)

    theList.remove(10)
    theList.remove(30)
    println(theList)
}
```

When you run the above Kotlin program, it will generate the following output:

```
[10, 20, 30, 40, 50]
[20, 40, 50]
```

Quiz Time (Interview & Exams Preparation)

Q 1 - Can we make a mutable Kotlin list as immutable?

A - Yes

B - No

Q 2 - We can add two or more lists and create a single list using + operator:

A - True

B - False

Q 2 - What does the Kotlin list get() method do?

A - It gets the list element from the given index.

B - It returns all the values of the list

C - There is no such method supported by list

D - None of the above